NPS-CS-07-022

NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

SecureCore Software Architecture:
Trusted Management Layer (TML)
Kernel Extension Module Integration Guide

by

David J. Shifflett
Paul C. Clark
Cynthia E. Irvine
Thuy D. Nguyen
Timothy M. Vidas
Timothy E. Levin

December 2007

Approved for public release; distribution is unlimited

NAVAL POSTGRADUATE SCHOOL
Monterey, California 93943-5000

Vice Admiral Daniel T. Oliver (Retired)
President

Leonard Ferrari
Provost

This material is based upon work supported by the National Science Foundation (NSF). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of that agency.

This report was prepared by:

David J. Shifflett
Research Associate

Paul C. Clark
Research Associate

Cynthia E. Irvine
Professor

Thuy D. Nguyen
Research Associate

Timothy M. Vidas
Research Associate

Timothy E. Levin
Research Associate Professor

Reviewed by:

Peter J. Denning, Chair
Department of Computer Science

Released by:

Dan C. Boger
Interim Associate Provost
and Dean of Research

REPORT DOCUMENTATION PAGE

Form approved

OMB No 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (Leave blank)	**2. REPORT DATE** 20 December 2007	**3. REPORT TYPE AND DATES COVERED** Research; September 2006 - December 2007

4. TITLE AND SUBTITLE

SecureCore Software Architecture: Trusted Management Layer (TML)
Kernel Extension Module Integration Guide

5. FUNDING

CNS-0430566

6. AUTHOR(S)

David J. Shifflett, Paul C. Clark, Cynthia E. Irvine, Thuy D. Nguyen, Timothy M. Vidas, and Timothy E. Levin

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Postgraduate School
Center for Information Systems Security Studies and Research (CISR)
1411 Cunningham Road, Monterey, CA 93943

8. PERFORMING ORGANIZATION REPORT NUMBER

NPS-CS-07-022

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

National Science Foundation (NSF)

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

This material is based upon work supported by the National Science Foundation (NSF). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF.

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words.)

A mobile computing device has more inherent risk than desktops or most other stationary computing devices. Such mobile devices are typically carried outside of a controlled physical environment, and they must communicate over an insecure medium. The risk is even greater if the data being stored, processed and transmitted by the mobile device is classified. The purpose of the SecureCore research project is to investigate fundamental architectural features required for the trusted operation of mobile computing devices so the security is built-in, transparent and flexible. A building block for the SecureCore project is a Least Privilege Separation Kernel (LPSK). The LPSK together with extension modules provides the security base. Integration of extension modules with the LPSK is described, including coding techniques, and compile and link directions.

14. SUBJECT TERMS

High Assurance, Security Kernel

15. NUMBER OF PAGES

18

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
Unclassified	Unclassified	Unclassified	Unclassified

This page left intentionally blank

NPS-CS-07-022

Trustworthy Commodity Computation
and Communication

SecureCore Technical Report

SecureCore Software Architecture:

Trusted Management Layer (TML)
Kernel Extension Module Integration Guide

David J. Shifflett, Paul C. Clark, Cynthia E. Irvine, Thuy D. Nguyen, Timothy M. Vidas, Timothy E. Levin

December 20, 2007

The Center for Information Systems Security Studies and Research

Acknowledgements

This material is based upon work supported by the National Science Foundation under Grant No. CNS-0430566 and CNS-0430598 with support from DARPA ATO. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation or of DARPA ATO.

Author Affiliation:

Center for Information Systems Security Studies and Research
Computer Science Department
Naval Postgraduate School
Monterey, California 93943

Table of Contents

[THIS PAGE IS INTENTIONALLY BLANK]

1 Introduction

1.1 Background

SecureCore is a research project funded by the National Science Foundation (NSF) to investigate the fundamental architectural features required for trustworthy operation of mobile computing devices such as smart cards, embedded controllers and hand-held computers. The goal is to provide secure processing and communication features for resource-constrained platforms, without compromise of performance, size, cost or energy consumption. In this environment, the security must also be built-in, transparent and flexible.

This document describes the requirements for building extension modules that may be incorporated into the Trusted Management Layer (TML), specifically the Least Privilege Separation Kernel (LPSK). The LPSK is composed of modules which are used as the building blocks of the kernel implementation. These modules are referred to as core kernel modules. Kernel extension modules are separate from the core LPSK modules, providing additional functionality.

A description of the SecureCore software architecture and definitions can be found elsewhere [1]. This document assumes the reader is familiar with the architecture and terminology of the SecureCore project.

2 Kernel extension modules

Kernel extension modules are functionality outside the LPSK that need to execute within the same environment (e.g. privilege level 0, PL0) as the LPSK. Kernel extension modules will be developed and maintained separately from the LPSK, and will be combined with the LPSK at build time, during the link phase.

2.1 Layering

Sound software engineering practices require that a system, such as the LPSK, be decomposed into individual modules, and that these modules be organized into ordered layers, such that modules can only depend upon (e.g. call into) other modules in lower layers. The goal of this design methodology is to avoid circular call sequences. The layering and modules are shown in Figure 1.

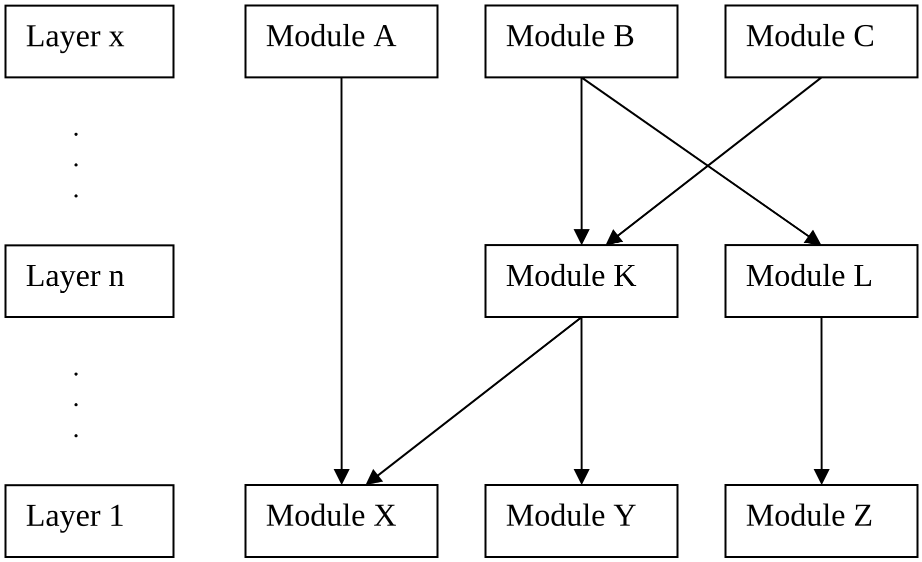

Figure 1. Modules and Layering

Modules A, B, and C, may call into all modules in lower layers, layers below ‘layer x’. Similarly, modules K, and L, may call into layers below ‘layer n’. Modules X, Y, and Z, are in the lowest layer and may not call into any other modules.

Kernel extension modules must be designed with this layering in mind. Kernel extension modules must be placed in a layer that is 1) above all the modules upon which they depend, and 2) below the modules that depend on the kernel extension module. This is shown in Figure 2.

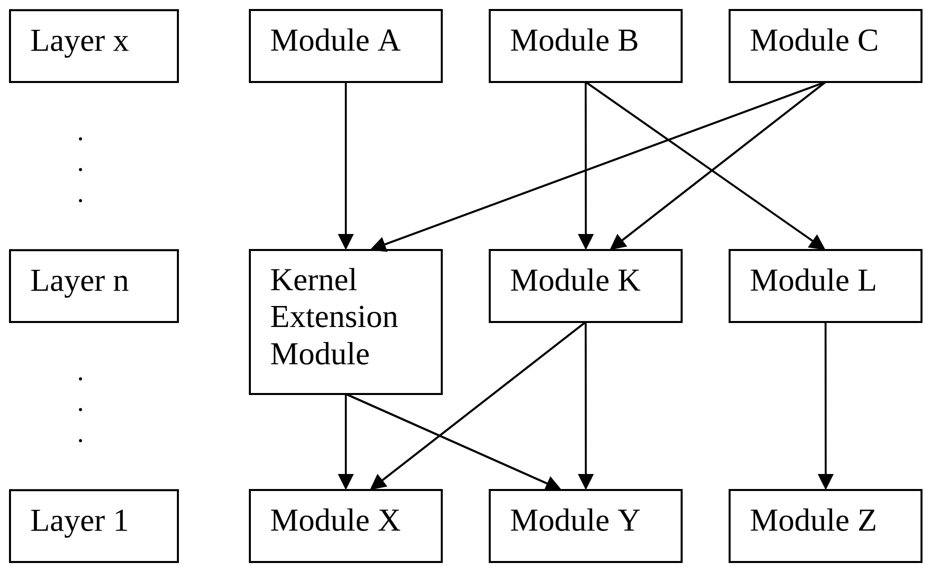

Figure 2. Layering of Kernel Extension Modules

2.2 Segment requirements

The code and data for kernel extension modules must be in segments separate from the core kernel modules. The stack segment will be shared by the core kernel modules and the kernel extension modules. The kernel extension modules must have at least one code segment, but may be composed of more than one code segment, and may have one or more data segments.

In the following examples the unique identifier for the kernel extension module, *kernel_mod_name*, does not have to be consistent across an entire kernel extension module. All code blocks with the same unique identifier will be combined into a single code segment and all data blocks with the same unique identifier will be combined into a single data segment.

2.2.1 Code segments

To place kernel extension module C-language code into its own segment, the code must be specified to be in a code segment other than the default code segment. This is accomplished by enclosing all code within a ‘pragma code_seg’ block, as follows:

#pragma code_seg ("*kernel_mod_name*" , "*kernel_mod_name*_CODE");

Kernel extension module code goes here
…

#pragma code_seg ();

where *kernel_mod_name* is replaced by a unique identifier for the kernel extension module.

More than one code segment can be created by having multiple ‘pragma code_seg’ blocks, each with a unique identifier.

For assembly language implementations, the following is used to place code within a specific code segment:

> _TEXT segment para public '*kernel_mod_name*_CODE'
>
> Kernel extension module code goes here
> …
>
> _TEXT ends

where *kernel_mod_name* is replaced by a unique identifier for the kernel extension module.

More than one code segment can be created by having multiple ‘segment/ends’ blocks, each with a unique identifier. To specify multiple code segments the ‘label’ (_TEXT in the above example) would have to be unique for each code segment (e.g. _TEXT01, _TEXT02).

2.2.2 Data Segments

Kernel extension module data must be put into a data segment separate from the core kernel module data, using the ‘-nd’ compile switch (see Compilation requirements below). Optionally, kernel extension module data may be placed into more than one data segment by enclosing all C data declarations within a ‘pragma data_seg’ block, as follows:

> #pragma data_seg ("*kernel_mod_name*");
>
> Kernel extension module data declarations go here
> …
>
> #pragma data_seg ();

where *kernel_mod_name* is replaced by a unique identifier for the kernel extension module.

More than one data segment can be created by having multiple ‘pragma data_seg’ blocks, each with a unique identifier.

For assembly language data declarations, the following is used to place data within a specific data segment:

_DATA segment para public '*kernel_mod_name*_DATA'

Kernel extension module data goes here
…

_DATA ends

where *kernel_mod_name* is replaced by a unique identifier for the kernel extension module.

More than one data segment can be created by having multiple 'segment/ends' blocks, each with a unique identifier. To specify multiple data segments the 'label' (_DATA in the above example) would have to be unique for each data segment (e.g. _DATA01, _DATA02).

2.3 Compilation requirements

The core kernel module data and constants must be separate from the kernel extension module data and constants. The Open Watcom compiler has a switch that enables this functionality. Kernel extension modules must be compiled with the same compiler switches as the core kernel modules with the following additional switch:

-nd=*kernel_mod_name*

where *kernel_mod_name* is replaced by a unique identifier for the kernel extension module.

The '–nd' switch causes the compiler to create a default data segment that is unique to the kernel extension module. This data segment will contain all data, constants, string literals, and uninitialized data associated with the kernel extension module.

2.4 Interfaces

The kernel extension modules must provide a header file, or files, declaring all functions within the kernel extension module that the core kernel modules are expected to call. Likewise, the core kernel modules will provide a header file, or files, declaring all functions that are exported to kernel extension modules.

2.5 Linking

The TML is linked using the Open Watcom linker. When linking the TML, the core kernel module objects must be first in the list of objects to link, followed by the kernel extension modules. The linking will combine all the object code (.o files) for the core kernel modules with the object code for the kernel extension modules to create the LPSK executable. The developers of the kernel extension modules do not need to be concerned with the linking step, but rather simply concentrate on providing the kernel extension module object files.

References

[1] Clark, Paul C., Irvine, Cynthia E., Levin, Timothy E., Nguyen, Thuy D., Vidas, Timothy M., "SecureCore Software Architecture: Trusted Path Application (TPA) Requirements", NPS-CS-07-001, December 2007.

Initial Distribution List

1. Defense Technical Information Center 2
   8725 John J. Kingman Rd., STE 0944
   Ft. Belvoir, VA 22060-6218

2. Dudley Knox Library, Code 013 2
   Naval Postgraduate School
   Monterey, CA 93943-5100

3. Research Office, Code 09 1
   Naval Postgraduate School
   Monterey, CA 93943-5138

4. Karl Levitt 1
   National Science Foundation
   4201 Wilson Blvd.
   Arlington, VA 22230

5. Lee Badger 1
   DARPA
   3701 Fairfax Drive
   Arlington, VA 22203

6. David J. Shifflett 1
   Code CS
   Department of Computer Science
   Naval Postgraduate School
   Monterey, CA 93943-5118

7. Paul C. Clark 1
   Code CS/Cp
   Department of Computer Science
   Naval Postgraduate School
   Monterey, CA 93943-5118

8. Cynthia E. Irvine 2
   Code CS/Ic
   Department of Computer Science
   Naval Postgraduate School
   Monterey, CA 93943-5118

9. Timothy E. Levin 1
Code CS/Tl
Department of Computer Science
Naval Postgraduate School
Monterey, CA 93943-5118

10. Thuy D. Nguyen 1
Code CS/Tn
Department of Computer Science
Naval Postgraduate School
Monterey, CA 93943-5118

11. Timothy M. Vidas 1
Code CS
Department of Computer Science
Naval Postgraduate School
Monterey, CA 93943-5118